Listy, część pierwsza

Na bieżących zajęciach omówione zostaną podstawowe wiadomości na temat struktur danych, zwanych *listami*. Podany zostanie przykładowy sposób na konstrukcję listy, jej wykorzystanie oraz usunięcie.

Lista jest strukturą danych, której zadaniem jest przechowywanie informacji wraz z zapamiętaniem kolejności ich wystąpienia. Wiąże się to z koniecznością przechowywania dla każdego elementu listy informacji, pozwalającej zidentyfikować element następny i/lub poprzedni. Pod tym względem listy dzielimy na jednokierunkowe i dwukierunkowe. Lista jednokierunkowa składa się z elementów, które zawierają wskazanie na element następny albo element poprzedni (tylko jeden z nich). Lista dwukierunkowa zawiera zarówno informację, który element jest poprzedni, jak również który element jest następny w kolejności.

Listy najczęściej realizowane są w oparciu o mechanizm dynamicznej alokacji pamięci. W takim przypadku naturalnym sposobem na „wskazywanie“ elementu poprzedniego, względnie następnego, jest użycie wskaźnika. Zawiera on adres odpowiedniego elementu – składnika listy.

Przykładowy program, używający list, zaprezentowany jest w poniższym listingu (plik *baza.cpp*)

```
#include <iostream>
#include <new>
using namespace std;

struct student
{
   char imie[20];
   char nazwisko[20];
   float ocena;

   student *nastepny;
};

student wczytaj_studenta();
student *dodaj_wpis(student opis, student *poczatek);
void wypisz_liste(student *poczatek);
void usun_liste(student *poczatek);

int main()
{
// Wskaznik do poczatku listy studentow
student *lista = NULL;

// Wynik pytania do uzytkownika
char odpowiedz;

   cout<<"Program utworzy liste studentow" << endl << endl;
   do
   {
                // Zapytaj, czy kontynuujemy wpisywanie
                do
                {
                   cout<<"Podajesz dane nowego studenta? [t/n]: ";
                   cin>>odpowiedz;
                   cin.ignore();
                }
                while( odpowiedz!='t' && odpowiedz!='n' );

                if(odpowiedz == 't')
                {
                student dodawany, *poczatek;
```

```
                dodawany = wczytaj_studenta();
                poczatek = dodaj_wpis(dodawany, lista);
                if(poczatek != NULL)
                        lista = poczatek;
                else
                        cout<<"Nie powiodlo sie dodanie studenta do bazy"<<endl;
            }
    }
    while(odpowiedz == 't');

    cout<<"Wypiszemy teraz zebrane dane:" << endl << endl;
    wypisz_liste(lista);

    cout<<"Na koniec nalezy usunac liste z pamieci" << endl;
    usun_liste(lista);
}

student wczytaj_studenta()
{
student wynik;

    cout<<"Podaj imie: ";
    cin.getline(wynik.imie, sizeof(wynik.imie));

    cout<<"Podaj nazwisko: ";
    cin.getline(wynik.nazwisko, sizeof(wynik.nazwisko));

    cout<<"Podaj ocene: ";
    cin>>wynik.ocena;
    cin.ignore();
    cout<<endl;

    wynik.nastepny = NULL;
    return wynik;
}

student *dodaj_wpis(student opis, student *poczatek)
{
student *nowy;

    // Tworzymy dynamicznie nowa strukture "student"
    try {
            nowy = new student;
    }
    catch(bad_alloc) {
            // W wypadku bledu alokacji funkcja zwraca wskaznik NULL
            return NULL;
    }

    // Kopiujemy dane do pamieci
    *nowy = opis;

    // Wstawiamy nowy element na poczatek listy
    nowy->nastepny = poczatek;

    // Zwracamy informacje, gdzie obecnie jest poczatek listy
    return nowy;
}

void wypisz_liste(student *poczatek)
{
     if(poczatek == NULL)
     {
       cout << "Lista jest pusta" << endl << endl;
```

```
                return;
    }
    else
    {
    unsigned long i = 1;
    student *aktualny;

      aktualny = poczatek;
                cout << "Lista studentow zawiera nastepujace wpisy:" << endl << endl;
                while(aktualny != NULL)
                {
                   // Wypisz pojedyncza strukture
                   cout << "***** Wpis numer " << i << " *****" << endl;
                   cout << "Imie: \t\t" << aktualny->imie << endl;
                   cout << "Nazwisko: \t" << aktualny->nazwisko << endl;
                   cout << "Ocena: \t\t" << aktualny->ocena << endl << endl;

                   // Zaktualizuj licznik rekordow bazy
                   i++;

                   // Przejscie do kolejnego elementu na liscie
                   aktualny = aktualny->nastepny;
                }
   }
}

void usun_liste(student *poczatek)
{
student *kasowany, *kolejny;

   // Dla pustej listy nie ma nic do robienia
   if(poczatek == NULL) return;

   // Zaczynajac od poczatku listy...
   kasowany = poczatek;
   while(kasowany != NULL)
   {
                // Zapamietaj, gdzie jest kolejny element - PRZED kasowaniem obecnego
                kolejny = kasowany->nastepny;

                // KASUJ rozwazany obecnie element listy
                delete kasowany;

                // W nastepnym kroku petli kasuj kolejny element
                kasowany = kolejny;
   }
}
```

Program używa struktury *student*, w której najważniejszy z punktu widzenia rozważanych zagadnień jest wskaźnik *następny*. Każda zmienna typu *student* przechowuje dane pojedynczej osoby oraz informację, gdzie w pamięci znajduje się opis kolejnego studenta (czyli kolejna struktura *student* na liście osób), zapisaną we wskaźniku *następny*. Powstaje pytanie: jaką wartość ma wskaźnik *następny* w strukturze, będącej ostatnim elementem na liście? Powszechnie stosowane jest wpisanie w takim przypadku wartości *NULL*, oznaczającej niepoprawny adres (nie wskazujący na żaden obiekt).

Najważniejsze pytania, dotyczące obsługi list są następujące: jak utworzyć listę oraz jak ją usunąć. Tworzenie listy polega na dodawaniu do niej pojedynczych elementów. Ponieważ każdy element listy zawiera informację, gdzie jest następny element, musimy zapamiętać dla listy jedynie wskaźnik do pierwszego elementu – to wystarczy, gdyż pozostałe elementy listy dostępne są poprzez jej przeglądanie po kolei. To determinuje metody tworzenia i usuwania list.

Zaprezentowany program tworzy listę poprzez dodawanie nowych elementów do listy na jej początek. Jest to metoda najprostsza, gdyż nie wymaga np. znalezienia ostatniego elementu na liście, co byłoby konieczne, aby dodawać elemnty na koniec listy. Wymaga jednak zmodyfikowania wskaźnika do pierwszego elementu listy, gdyż każdy poprawnie dodany element stanowi od tej pory poczatek listy. Modyfikacja dokonywana jest w funkcji głównej programu, na podstawie wskaźnika zwracanego przez funkcję *dodaj_wpis*, dodającą nowy element do listy. Aby dodać element na początek listy należy:

1. Zaalokować dynamicznie pamięć na nowy element listy operatorem *new*. Jeśli się to nie powiodło, funkcja *dodaj_wpis* zwraca wskaźnik *NULL*.
2. Jeśli alokacja pamięci zakończyła się sukcesem, można utworzyć wpisy w otrzymanym obszarze pamięci. Dane tworzymy według uznania (w funkcji *dodaj_wpis* dane kopiowane są ze struktury, przekazanej jako argument funkcji), wskaźnik na kolejny element ustawiamy na poprzednią wartość wskaźnika początku listy – dotychczasowy pierwszy element listy będzie teraz drugim elementem.
3. Zmienić wskaźnik początku listy na adres nowo utworzonego elementu listy tj. adres otrzymany w wyniku działania operatora *new*, o ile powiodło się utworzenie nowego elementu.

Funkcja *dodaj_wpis* realizuje dwa pierwsze z powyższych zadań. Jako swój wynik przekazuje nową wartość wskaźnika do pierwszego elementu listy lub *NULL,* w wypadku niepowodzenia podczas tworzenia nowego elementu listy – jeśli funkcja *dodaj_wpis* zwróciła wartość różną od NULL, to należy tę wartość przypisać do wskaźnika początku listy.

Aby usunąć listę, celem zwolnienia pamięci zaalokowanej w toku jej tworzenia, należy:

- pobrać adres początkowego elementu listy do zmiennej wskaźnikowej *kasowany*, której zadaniem będzie przechowywanie adresu obecnie usuwanego elementu listy
- dopóki wskaźnik *kasowany* ma wartość różną od *NULL*
    1. zapamiętać wartość wskaźnika do następnego elementu listy w zmiennej *następny*
    2. zwolnić pamięć, w której przechowywany jest aktualny element listy (*delete kasowany;*)
    3. wykonać przypisanie *kasowany=następny;* powodujące przejście do usuwania kolejnego elementu w następnym kroku pętli

Przykładowa realizacja powyższego algorytmu zawarta jest w funkcji *usun_liste*.

Zagadnienia do rozważenia

1. Jak skonstruować rekurencyjne usuwanie listy?
2. Jak należy napisać funkcję, dodającą element do listy, aby nie było konieczne zmienianie wskaźnika początku listy na zewnątrz tej funkcji?